DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION
DSTO

Issues Concerning the Development of a CORBA Trading Service

Andrew K. Lui and David J. Miron

DSTO-TR-0841

Issues Concerning the Development of a CORBA Trading Service

Andrew K Lui and David J Miron

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-TR-0841

ABSTRACT

A core feature of the Experimental Command Communication Control Intelligence Technology Environment (EXC3ITE) is its support for distributed applications. A key feature of distributed applications is that the components of the applications can be developed and managed in a distributed manner. At run time, the applications are dynamically composed using the components distributed in a networked environment. The Location Service is a critical enabling technology of distributed applications. It allows components to discover other collaborative components during run time. The Trading Service is a more advanced variant of the Location Service. This work focuses on the design and development of a trader running on CORBA. In particular an implementation of a trader using Java is presented, in addition to discussions on development issues and future research directions on traders.

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE & TECHNOLOGY ORGANISATION DSTO

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury South Australia 5108 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567
© Commonwealth of Australia 1999
AR-011-019
July 1999

Issues Concerning the Development of a Trading Service

Executive Summary

The Experimental Command Communication Control Intelligence Technology Environment (EXC3ITE) provides a platform for the Defence Science and Technology Organisation (DSTO) and its industry partners to demonstrate future technologies and concepts.

A core feature of EXC3ITE is the use of distributed and component-based software, where a component is a remote software entity that is dynamically linked to a distributed application at run time. Each component is a functional unit in the application, cooperating with other components. It is envisaged that a distributed environment will be populated by components developed by teams within the DSTO and commercial-off-the-shelf (COTS) software. Example components being developed include the Image Management and Dissemination project (IMAD) and the Geospatial Services Segment (GSS) in Information Technology Division (ITD). An important edge of such component-based architecture is the possibility of integrating components from different systems into novel applications.

The trader is a key enabling service supporting the dynamic integration of components into applications. The trader is a kind of registry where an individual component can signal its presence in the distributed environment. So cooperative components in an application can use the trader to discover other components, and then to make remote invocations.

In this work a CORBA Trading service is developed and deployed. The trader, called JavaTrader, is itself written in Java with the work driven by the requirements of the IMAD project. Using Java as the implementation language gives the trader platform independence. The capability to run on a range of platforms is important in a distributed computing environment like EXC3ITE. The outcome of this work is twofold. The portable trader will not only become an important service in the EXC3ITE environment, but also provide us with a platform on which further research on trading can take place.

Authors

Andrew K Lui
Information Technology Division

Dr. Andrew Lui BSc(Syd) PhD(ANU) is a Research Scientist in the C3 Information Systems Concepts Group. He joined DSTO in 1998 after working on a computer aided teaching project for a year in ADFA. He is currently working in the Imagery Management and Dissemination (IMAD) project. His research interests include distributed computing and architecture, content based image retrieval and multimedia systems.

David J Miron

Information Technology Division

Dr. David Miron B App Sc Maths (RMIT) M AppSc (Monash) PhD(ANU) was a Research Scientist in the C3 Information Systems Concepts Group during the course of this work. He joined DSTO in 1997 after completing his PhD in Parallel Numerical Algorithms. David is now a Lecturer in the School of Mathematical and Computer Sciences at the University of New England. His current research area is in the development of online teaching tools.

Contents

1. Introduction

The Experimental Command Communication Control Intelligence Technology Environment (EXC3ITE) provides a platform for the Defence Science and Technology Organisation (DSTO) and its industry partners to demonstrate future technologies and concepts.

A core feature of EXC3ITE is the use of distributed and component-based software, where a component is a remote software entity that is dynamically linked to a distributed application at run time. Each component is a functional unit in the application, cooperating with other components. It is envisaged that a distributed environment will be populated by components developed by teams within the DSTO and commercial-off-the-shelf (COTS) software. Example components being developed include the Image Management and Dissemination project (IMAD) [Grigg et al. 1999] and the Geospatial Services Segment (GSS). An important edge of such component-based architecture is the possibility of integrating components from different systems into novel applications.

The trader is a key enabling service supporting the dynamic integration of components into applications. The trader is a kind of registry where an individual component can reveal its presence in the distributed environment. As such cooperative components in an application can use the trader to discover other components, and hence to make remote invocations on those components.

In this work a Common Object Request Broker Architecture (CORBA) Trading service has been developed and deployed. The trader itself was written in Java with the work driven by the requirements of the IMAD project. Using Java as the implementation language allows the trader to be platform independent. The capability to run on a range of platforms is important in a heterogeneous distributed computing environment like EXC3ITE. The outcome of this work is two-fold. The portable trader will not only become an important service in the EXC3ITE environment, but also provide us with a platform on which further research on trading can take place.

This report is organised as follows. In Section 2 a background and description of component-based architecture technologies in distributed environments will be given. The basics of CORBA will be discussed. In particular we will focus on the role of location services in the dynamic integration of components. Differences between the Naming Service and the Trading Service will also be briefly discussed.

In Section 3, we will discuss several important issues concerning the design and implementation of a trader based on Java. An overview of the design and detailed discussion on some critical components will be given. In Section 4 a discussion is provided based on the experience gathered from the design and implementation of the trader. This discussion is based upon several research issues involving the Trading Service. A summary of the report is given in Section 5.

2. Background

2.1 Component Based Architecture Technologies

A component based software architecture supports the 3-tier architecture in which applications, business processes and data repositories are integrated. A 3-tier architecture is illustrated in Figure 1.

Some of the key technologies that support component based architectures are the Common Object Request Broker Architecture (CORBA) from the Object Management Group (OMG) [OMG 1998], the Distributed Component Object Model (DCOM) from MicroSoft and the Remote Method Invocation (RMI) used in Java. These technologies provide the necessary services to support the business process in the 3-tier architectures, thus allowing distributed components to work together in applications.

Of particular interest to this work is CORBA due to its superiority in a number of aspects. CORBA runs on all major operating systems including Unix, NT, Windows 95, MacOS and Java OS. CORBA also provides interfaces to a number of major modern

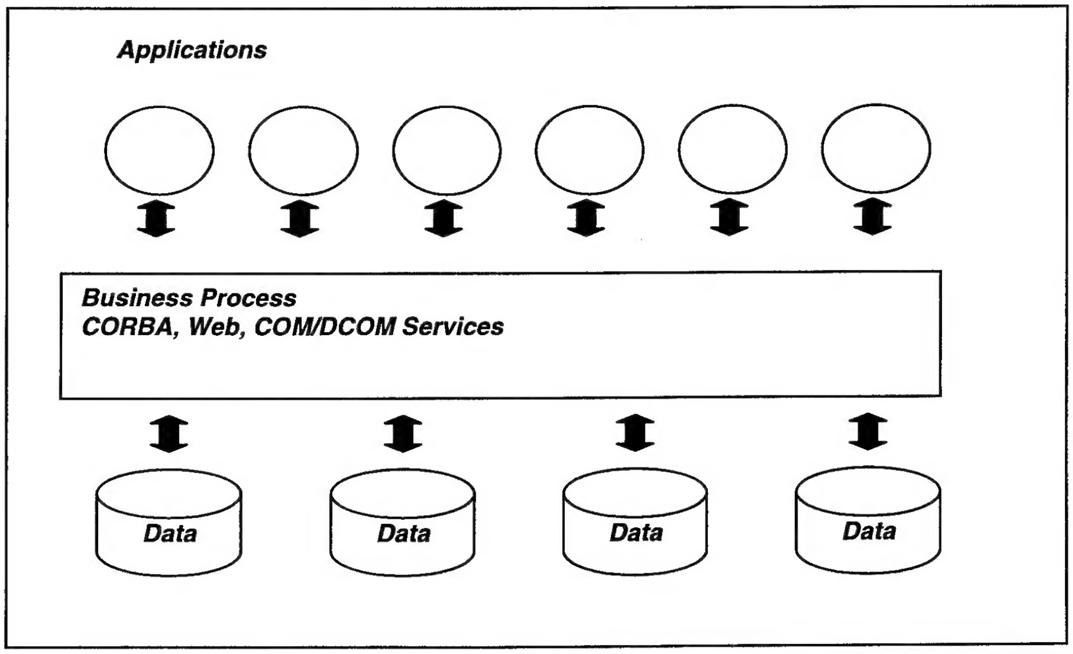

Figure 1: Three-tier architecture in a distributed environment.

languages including Java, C and C++. A CORBA based system is also highly scalable. The core pipeline of CORBA is a loosely coupled federation of distributed Object Request Brokers (ORBs). The pipeline is extended to a computer host simply by installing an ORB.

CORBA and Java form a very important partnership in supporting true platform and operating system independence. A CORBA environment can reside on any computer that has a Java Virtual Machine installed along with a network connection. In the future, Java's own RMI will interoperate with CORBA. Thus, the underlying Internet Inter-ORB Protocol (IIOP) of CORBA will become the backbone of RMI.

2.2 Building Distributed Applications in CORBA

In a CORBA environment, applications are composed of interacting distributed components (Figure 2). The interaction is in the form of method invocation on the CORBA objects of remote components. Publishing a component in CORBA requires that its interface be expressed using the Interface Definition Language (IDL). The IDL exposes the methods of the CORBA objects of a component as a set of Application Program Interfaces (APIs). Once a component is exposed, other components can locate

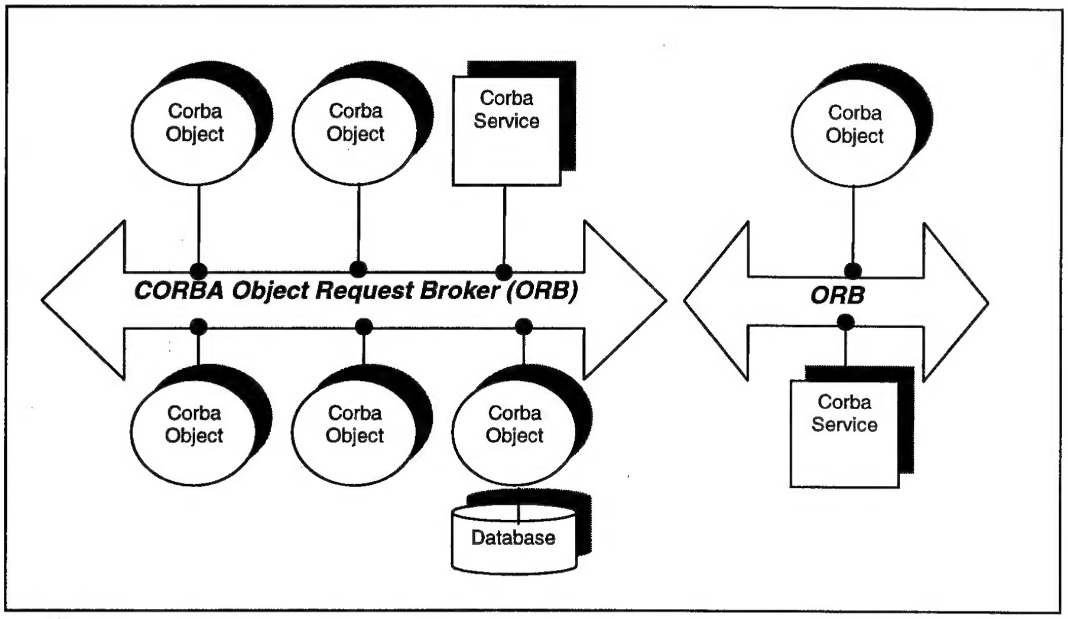

Figure 2. A typical CORBA distributed environment. The environment consists of a number of CORBA objects connected to the ORB and a number of CORBA standard services. Some of the CORBA objects can be 'wrappers' which provide a CORBA interface for other components like a database. The backbone ORB is highly scalable. Once a local environment is installed with an ORB, it is logically connected to other CORBA environments.

and invoke methods on the CORBA objects of the component using its Interoperable Object Reference (IOR).

The IOR contains all the necessary information for a client component to bind to a server component. For the components of an application to interact, one needs to obtain the other's IORs so that method invocations can be made. This problem can be resolved in two different ways. The more direct but inflexible way is to hardcode all the IORs of target components into client components. On the other hand, the CORBA Location services provide a much more flexible approach in that components and their IORs can be discovered dynamically.

2.3 The CORBA Location Services

In a distributed environment many server components are available to clients wanting to make use of them. Rather than informing every client of its IOR, a server component registers itself with a CORBA Location service. The CORBA Location service thus acts as a central repository of IORs of server components. Clients can query the repository for the existence of an appropriate component and be returned the IOR allowing a connection to be made. This process is illustrated in Figure 3.

The CORBA Location services consist of the Naming service and the Trading service. These two services differ significantly in a number of aspects, which are briefly described in the following two subsections.

2.3.1 The Naming Service

The Naming Service can be conceptualised as a white pages telephone directory. In a white pages, phone numbers can be found for a specific person using their registered names. The phone numbers can then be applied to make a phone connection.

Similarly, components are registered with a Naming Service by component name. Client applications that wish to make use of a component contact the Naming Service and ask for a component by its name. The Naming Service then returns the IOR of the component back to the client.

2.3.2 The Trading Service

The Trading Service is similar in concept to a yellow pages telephone directory. Unlike a white pages, a yellow pages lists contact information by category rather than name.

Similar to the Naming Service, a trader accepts registrations from components and groups them by service type. Clients can then query a trader for components of a particular service type. From the results of this query a client is free to choose whichever components best meets their need.

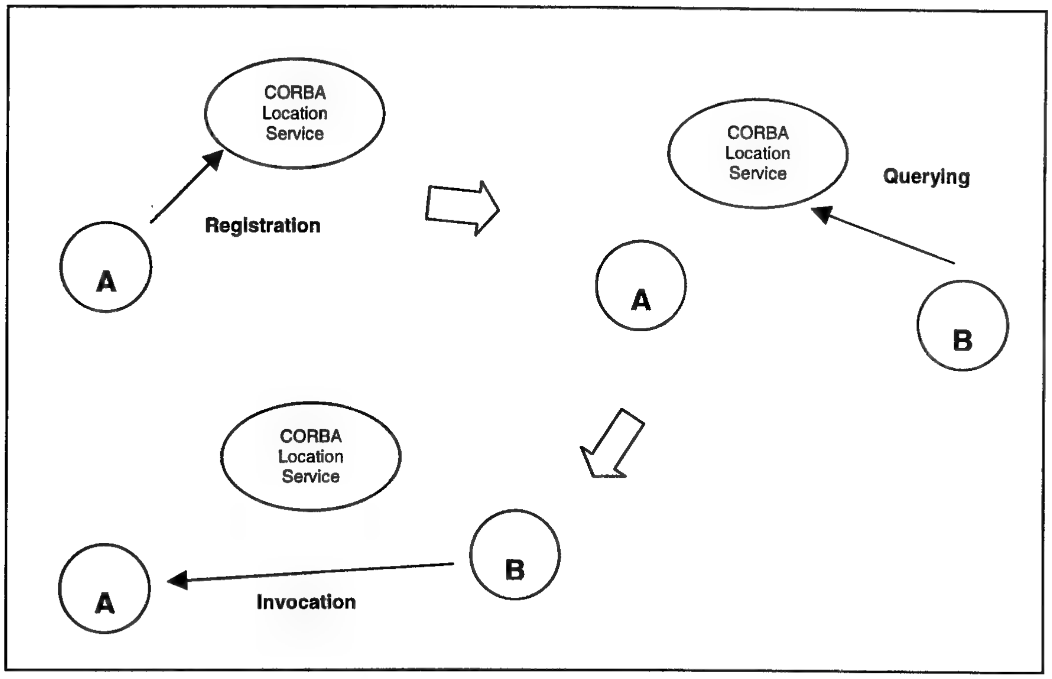

Figure 3. Component discovery using the CORBA location services. The component A wanting to publish itself first registers with the location service. The component B wishes to use the service offered by A. B then queries the location service and the location service returns the IOR of A to B. B then makes a method invocation on A.

To assist clients selecting the appropriate server component, the Trading Service also stores the properties of the server components. These properties describe the characteristics of the server components. The clients can then make an informed decision when selecting a server component. The Trading service assumes that the server components provide their true characteristics.

2.4 Traders

The CORBA Trading service offers greater flexibility in component registration and discovery than the Naming service. The downside is the added complexity in the usage and the design of the Trading service.

Before a component can register with a CORBA trader, the trader must have been set up with appropriate service types. Recall that a component registration consists of its service type and its properties. Thus the trader must first know about the service types and the properties associated with the service types. A service type is a string with an arbitrary number of associated typed properties. For example, a service type *Printer* may have the properties of *ppm* (page per minute) of type integer and *printcolour* of type boolean.

Once the trader is initialised with appropriate service types, server components can register, or *export*, their offers. A server component passes on to the trader its IOR and the properties settings that describe it.

Clients can then lookup, or *query*, the trader for the components they desire. After a client queries a trader for a particular service type, the trader will return a list of matching components. The client may specialise the query by specifying further criteria based on the properties of the components. The criteria are put in a special language similar to the form of a SQL query. For example, a client may seek for a fast colour printer by using the criteria "ppm > 10 and printcolour = true". The client may also ask the trader to return the matching components in some order. For example, the trader may be asked to return the list of printers in the order of decreasing ppm.

A CORBA trader manages the usage and design complexity by defining a number of functional interfaces. The Service Type Repository interface supports the addition, deletion and modification of service types and their associated properties. The registration and withdrawal of component offers is done through the Register interface. The Lookup interface allows clients to query for registered components of a service type and satisfying certain criteria. Trader administration tasks such as setting operational parameters can be done through the Admin interface.

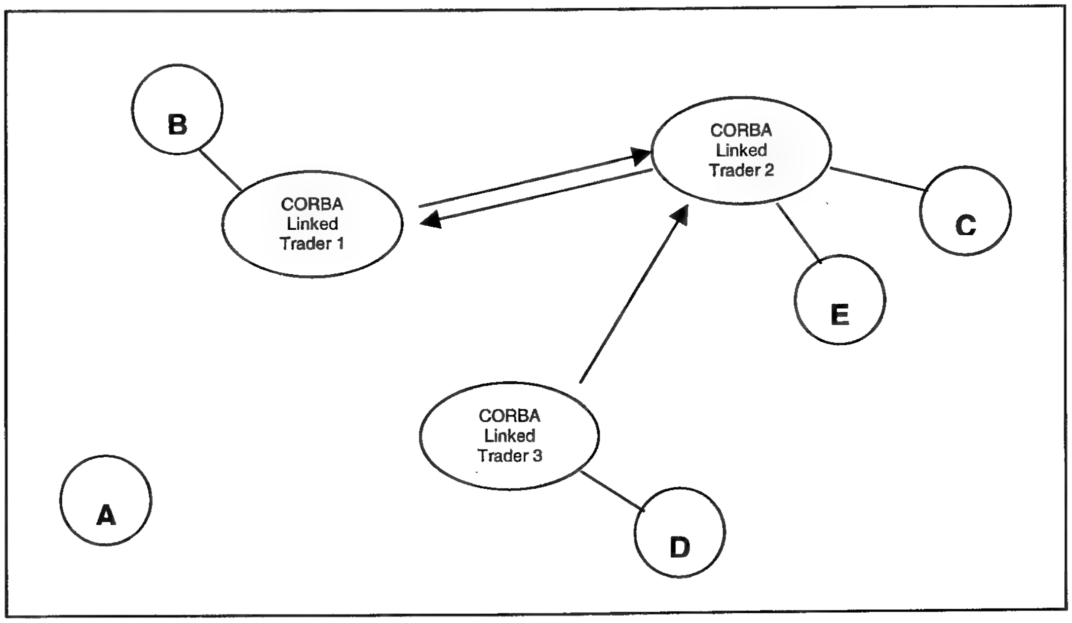

Figure 4: Federated traders. The arrows indicate directional links between traders. A number of components are registered with different traders (indicated by a line). A is the component wanting to discover other components. If A queries trader 3, components B, C, D, E can all be discovered. However, if A queries trader 1, only B, C and E can be discovered because of the directional link between 2 and 3 is missing.

In a distributed environment with thousands of components, CORBA traders can be federated so that each local trader can serve a manageable number of components (Figure 4). The Link interface supports the establishment of links between traders. Queries received at a local trader can be passed onto other linked traders and the results of the queries then propagated back to the original client. Such links can be bi-directional or uni-directional. A number of operational parameters can be used to control the behaviour of federated queries. For example, one can control a trader in such a way that a query is always passed onto linked traders, never passed onto others, or only passed onto others if no matching components is found in the local trader. The federation of traders makes the CORBA trading service highly scalable.

The CORBA 2.2 standard defines various types of traders depending upon the interfaces that are implemented [ISO 1997]. The interfaces implemented by a query trader, a simple trader, a standalone trader, a linked trader, a proxy trader and a full trader are illustrated in Table 1.

Table 1: Different types of traders, as specified in ODP Trading Standard.

Interfaces / **Trader Type**	**Lookup**	**Register**	**Admin**	**Proxy**	**Link**
Query Trader	Y				
Simple Trader	Y	Y			
Standalone Trader	Y	Y	Y		
Linked Trader	Y	Y	Y		Y
Proxy Trader	Y	Y	Y	Y	
Full Trader	Y	Y	Y	Y	Y

3. Design and Implementation

To facilitate further trader research and to support ongoing EXC3ITE projects, a trader satisfying the following requirements is required:

1. Compliance to the OMG trading standard

The Open Distributed Processing (ODP) Trading Function Standard is a part of the CORBA 2.2 specification and specifies the computational interfaces for a trader. However, the standard leaves room for various implementation approaches. The advantage of an ODP compliance trader is that the trader can seamlessly replace other ODP based traders. The client components are not affected, thus reducing the impact of deployment in the EXC3ITE environment. Another advantage is that the trader will have no problem federating with other ODP based traders.

2. Platform portability

Many distributed environments, such as EXC3ITE, often comprise a various hardware platforms and operating systems. To make the trader portable and able to serve from a large number of hosts, a Java implementation is desired. The basic requirement of running a Java program is a Java Virtual Machine (JVM), which is now available in almost every major platform.

3. Availability of source code and design details

Another main objective of this work is to enable us to identify and pursue on a number of important research issues in resource discovery in distributed computing environment. Building a trader provided us with experience in design and the source code which can form the basis of development of further trader prototypes.

3.1 Design Overview

In this section we will briefly describe the design and implementation of a *Linked Trader*. The ODP Trading Function Standard defines a *Linked Trader* as supporting service type management, service registration, service lookup, trader administration and trader federation. The *Linked Trader*, called JavaTrader, uses Java as the implementation language.

JavaTrader is based on the object oriented design paradigm. Object oriented systems generally promote efficient implementation and high reusability of components. More importantly object oriented systems are resilient to change, thus making functionality improvement easier.

A schematic diagram of JavaTrader is illustrated in Figure 5. The design and implementation of the five interfaces of a *Linked Trader*, namely the *Service Type Repository*, *Register Interface*, *Lookup Interface*, *Admin Interface* and *Link Interface*, is the focus of this work. An analysis of the problem has identified that the following components are also required so the functionality of the five interfaces is supported.

1. Data Repository for storing trading information.

Traders need to store several types of information, including registered service types, registered service offers and link information. Storing such trading information persistently ensures that the traders can recover from various problems. A suitable data repository is required which allows the adding, storing and querying of trading information.

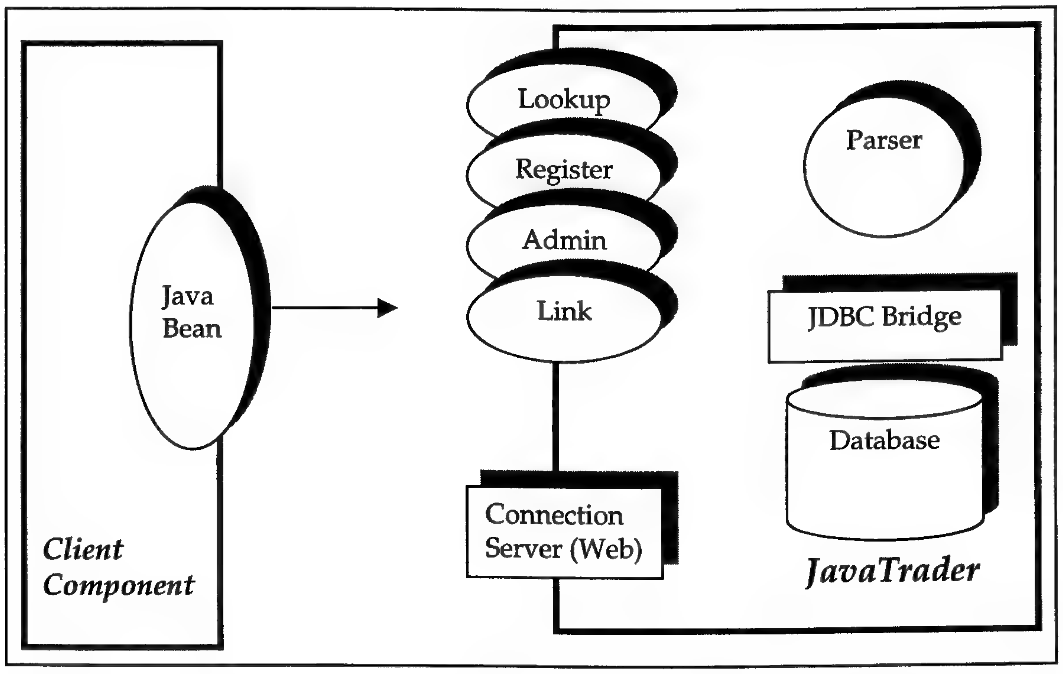

Figure 5. A schematic diagram of JavaTrader. JavaTrader exposes four interfaces (sets of APIs) to clients: Lookup, Register, Admin and Link. The trader's IOR is served through the Connection Server, so that clients can make initial connection to the trader. The trader has a parser for converting trader queries into SQL queries. A database is used to store trading information and it is accessed through a JDBC bridge. On the client side, JavaTrader provides a JavaBean component for easier use of the trader.

2. Parser for parsing queries.

The language used to query traders is defined in the Trading Function Standard. Queries received at the Lookup interface need to be parsed and possibly converted to other forms suitable for further evaluation.

3. Configuration Manager for managing trader configuration.

The ODP Trading Function Standard has defined a large number of parameters through which traders can be configured. Most of these parameters can be adjusted through the Admin interface and such changes are desired to be permanent. A configuration manager is required that these parameters can be stored persistently.

4. Connection Server for publishing the trader's IOR.

A client wanting to connect to a trader must first hold the trader's IOR. Apparently we have returned to the original problem of how to discover other components in the distributed environment. But in fact the dimension of the problem is greatly reduced in that the client only needs to know one IOR (the trader's) in order to obtain other IORs.

Instead of hard coding a trader's IOR into every client component, the IOR is made publicly available through a *well-known location*. A Connection Server is designed to serve a trader's IOR using HTTP. A client can obtain a trader's IOR through *a well-known* URL. The added level of indirection makes the mechanism of trader connection flexible.

5. JavaBean components for easy client access to the trader.

A transaction with a trader typically involves a number of CORBA method invocations. Each method invocation may require a number of steps including connection, parameter preparation, returning value handling and exception handling. Coding the client can require significant amount of effort.

A set of client side JavaBean components are designed to make coding trader transactions easier. The JavaBean components shield off the CORBA method invocations from the client. The client can, for example, query a trader by instantiating a TraderQuery JavaBean, initialising its properties and activating the JavaBean.

3.2 Detailed Design

3.2.1 Configuration of a Database for storing trading information

JavaTrader uses a SQL database to store trading information including service types, service offers and link information. A SQL database can readily support the requirement of inserting, deleting and querying trading information. Submitting SQL scripts to the database allows the manipulation of trading information.

The Java DataBase Connectivity (JDBC) bridge was used as the interface to the SQL database. JDBC is a standard Java Application Programming Interface (API) allowing Java components to access any JDBC compliance database. Most major commercial database systems support a JDBC interface. However, a Java implementation of such a database system is more attractive because this allows a pure Java trader and achieves true platform independence. InstantDB is an example of a Java implementation of a JDBC/SQL database [InstantDB 1999].

Trading information, such as the properties of a service offers, has a defined datatype specified by the ODP Trading Function Standard. The datatype specified is the CORBA type *Any* which is a union structure of various primitive types such as integer, character and string. To store a CORBA Any value in a SQL database requires that the *Any* value be type-cast to an equivalent datatype in the database. Conversely, to obtain a value from the database requires that the value be type-cast back to its original *Any* type. However, the mapping between these datatypes is neither always possible nor reversible. Some constraints are introduced to the permissible range of datatypes (see Table 2).

Table 2: Mapping between JavaTrader's datatype and the underlying DataBase representation.

CORBA Any Type	**SQL DataBase Type**
TCKind.tk_short	INT
TCKind.tk_ushort	INT
TCKind.tk_long	LONG
TCKind.tk_ulong	LONG
TCKind.tk_float	REAL
TCKind.tk_double	DOUBLE
TCKind.tk_longdouble	DOUBLE
TCKind.tk_boolean	CHAR(n)
TCKind.tk_char	CHAR
TCKind.tk_octet	BINARY
TCKind.tk_string	CHAR(n)

3.2.2 Federated trading management

JavaTrader supports the federation of multiple traders so that trading information can be shared. Each of the traders not only processes queries submitted by objects of the local domain but also by other linked traders. Effectively, a query received by a trader is recursively submitted to other linked traders. The query results obtained from all the queried traders are aggregated and returned to the client. The links between traders are uni-directional. Figure 4 illustrates that a query submitted to Trader 3 can be submitted to Trader 2 and Trader 1 recursively.

The workload of federated trading management is shared between the *Link* and the *Lookup* interfaces. The *Link* interface is responsible for the addition, deletion and modification of links of a trader. The links are information regarding the linked trader's IOR and policy of link traversal. For example, the policy of a link can be specified so that a query is submitted to the linked trader only if the local trader cannot offer a matched service offer. Such link information is then used by the *Lookup* interface in query processing. In addition to evaluating queries with local trading information, the Lookup interface manages the submission of the queries to linked traders and gathers the results.

Implementing federated trading presents difficulties because the operations involved are highly dynamic and distributed. Some of the problems that can occur are runaway queries and unproductive revisiting of a trader. By managing the mobility of queries both of these problems can be solved. The ODP Trading Standard specifies two trader configuration parameters for such purpose: *hop count* and *requestID*. Both parameters are passed along with every query.

The first parameter *hop count* is designed to remove the possibility of runaway queries. This parameter specifies the maximum search depth in the graph of federated traders. For example, a hop count of one in the previous example allows the recursive query of Trader 3 only. However, a hop count of two will include Trader 2 and Trader 5 in the search.

The second parameter *requestID* is designed to prevent the revisiting of a trader in a recursive query. This parameter contains the identity of the initiator of a recursive query. A trader can acknowledge if the query is initiated by itself and it can discard the query accordingly.

3.2.3 System integration and performance

We have encountered a number of problems in system integration involving Java and CORBA. The first problem is caused by different handling of null values in the ORBs. Java implementations of an ORB disallow passing null values in structure parameters. The marshalling or unmarshalling of null values in structures will raise an exception. Instead arrays of length zero (eg. String[0]) should be used to indicate a *null value*. However, C implementations of ORB allow and in fact prefer the use of null. So problems will arise when a Java CORBA object links with a C CORBA object. For example, this problem occurs when JavaTrader federates with a C implementation of a trader. This indicates that CORBA full portability is yet to be achieved.

The use of the CORBA datatype *any* also causes difficulties. The *any* datatype is a generic type capable of storing a number of data primitives ranging from integers to strings. This type facilitates the passing of complex data structure through an ORB. However, the creation and the assignment of an *any* variable involve a tricky manipulation. Coding effort is significantly increased as a result.

Compared to other implementations of trader, the use of Java as the implementation language incurs a small performance penalty. However, the performance gap is rapidly decreasing between the use of Java and other languages such as C. A number of Java Virtual Machines (JVM) are now implemented with Just-in-Time (JIT) run time compilers. The JIT compilers convert Java bytecode into native machine code as classes are loaded into the JVM. The result is a slower initialisation time (for the compilation), but a substantial increase in performance during execution. Persistent applications such as traders will benefit from this because they rarely require initialisation.

4. Discussion

One of the goals of JavaTrader is to provide an experimental test-bed on which further research prototyping can take place. The operational experience gained and a survey of the literature has enabled us to identify a number of important research issues.

4.1 Federation of Traders

4.1.1 Model of query processing

The current model of recursive query handling in federated traders has other, possibly better alternatives. The current model processes a query by passing the query onto other linked traders. The source trader then waits for all the recursive queries to finish. It then gathers the results, and finally returns these to the original requestor. The original requestor has no knowledge of the status of the query processing once the query is submitted. The query may follow all the links to visit all reachable traders. On the other hand, arbitrary linkage also makes the revisit of a trader possible. The use of the query parameter requestID can help prevent the revisiting of the source trader. However, the query can still revisit other non-source traders, causing redundant processing to occur.

An alternative approach is to use iterative query handling. The source trader, upon receiving a query, then plans out the routing of the query to other linked traders. The routing plan is based on the link topology of all the reachable traders. Such link topology of the traders can be obtained by progressively querying the Link interface of the traders, one after the other. Then for all the traders within the distance specified by the query parameter *hop count*, the source launches a query and aggregates the results. This alternative approach ensures that each trader processes a query only once.

4.1.2 Distributed traders and interworking traders

There are other research activities addressing issues on a broader level of trader federation. Richman and Hoang proposed that there are two distinctive models of trader federation [Richman and Hoang 1995a]. *Distributed Traders* operate within an organisation with uniform service types and policy rules. The ODP Trader is an example of distributed traders. This model has a limitation when trader federation spans across organisation boundaries. The second model of *Interworking Traders* deals with the interaction and exchange of trading information across organisational boundaries. The key issue is to design a mechanism so that conversion of trading information (such as service type identifiers) can be carried out between two trading domains.

4.2 Fault Tolerance and Quality of Service

In real life traders often offer advice to their clients on the quality of services available from a service provider. However, clients of ODP Traders have no information about the quality of the service offers obtained. For example, a client may find a component obtained from the trader is no longer available, causing the connection to fail. In another example, a client may experience a very slow response from one component, but a prompt response from another component of the same type. One approach to solve these problems is to augment the ODP Trader with the ability to offer advice on the quality of service of the service providers.

Offering advice on the quality of service enables clients to make informed decisions on selecting a server component. Example types of quality of service include the connection uptime (percentage of the period of a component being ready) and the response time (time taken for results to return to a client). A trader can gather such information by auditing the server components' performance and client's feedback. Network monitoring services such as transmission availability forecast services can provide traders with information about network conditions [Blair and Jana 1999].

4.3 Unified Resource Discovery Repository with Diverse Access

Traders can be regarded as a resource discovery tools in a distributed environment. Other resource discovery tools such as the Naming Service and the Directory Service are found to share a similar design. Typically, these tools have an underlying repository for holding persistent information about the resource available in the distributed environment. Also, they provide a query or a search engine for obtaining resource information. Therefore, it is not surprising that traders can be implemented using a directory service such as X.500. Some work has reported on the design of such systems and additional components required for functionality such as trader federation [Kutvonen 1997][Richman and Hoang 1995b][Waugh and Bearman 1995].

The possibility of implementing ODP Traders (and also the Naming Service) on X.500 has allowed all these resource discovery tools to share the same information repository. The interfaces provided by the ODP Traders, the Naming Service and the Directory Service offer a diverse access to the same pool of information (see Figure 6).

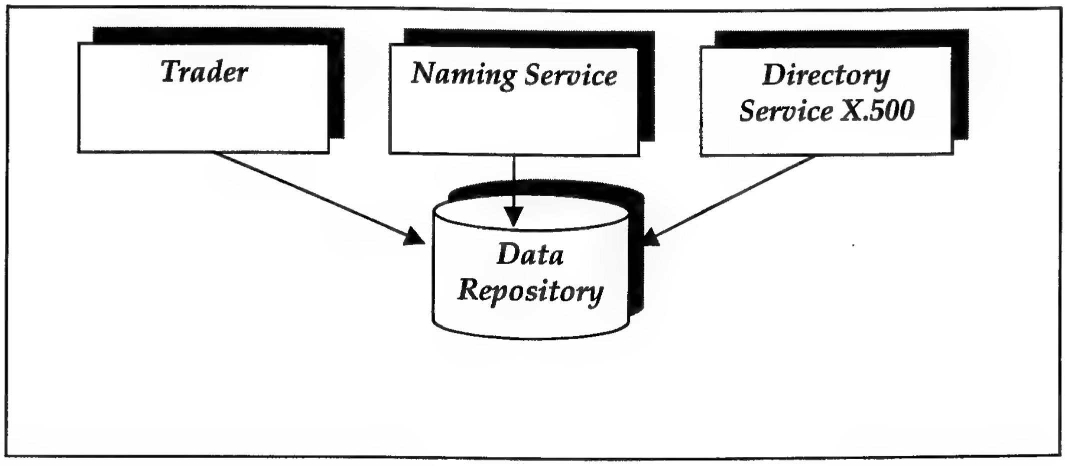

Figure 6: Potentially the trader, the naming service and the directory service can share the same data repository, thus providing diverse access with a unified resource.

5. Summary

This work began with the acknowledgement of the advantages of dynamic application integration in a distributed environment like EXC3ITE. Such application integration requires the ability of dynamically discovering cooperating components in the environment. We focused on the CORBA Location Services that play the role of a matchmaker of components so that applications can be integrated. The two types of Location Services, namely the Naming Service and the Trading Service, were then investigated.

The core of this work was on the development of a Java based Trading Service for EXC3ITE. The EXC3ITE environment is to be populated with a range of services such as imagery services by the IMAD project and geographical information services by the GSS project. The Trading Service is superior to the Naming Service in this environment because client components can discover services by their service types. The client does not need knowledge about the specifics of server components. However in the case of using the Naming Service, the client component needs to have prior knowledge of the name of the server component it wishes to connect. Traders offer application developers in EXC3ITE greater flexibility in designing their applications.

The design and implementation of the Java based Trader, called JavaTrader, were based upon the requirements of compliance to CORBA ODP standard, portability and source code availability. The Java Trader has been implemented as a Link Trader which contains the interfaces of Lookup, Register, Link and Admin. The trader stores trading information in a SQL database. A JDBC bridge is used as the interface between the interfaces and the database to ensure portability. In order to help clients making initial connection to JavaTrader, a Connection Server is designed to serve a JavaTrader's IOR through HTTP. Also JavaTrader supports client side development by providing a number of JavaBean components through which client component can easily access a JavaTrader.

JavaTrader has been successfully deployed in EXC3ITE, supporting the IMAD project. Potentially the trader will serve other EXC3ITE services in the near future.

The work concluded with a suggestion on future research direction on traders. The suggestion is based on experience gathered from the implementation and also literature survey.

6. Bibliography

Blair W.D. and Jana R. (1999), "A Transmission Availability Forecast Service for Internet Protocol Networks", DSTO Research Report DSTO-RR-0146, DSTO, 1999.

Grigg M. et al. (1999), "Component Based Architecture for a Distributed Imagery Library System", in Proceedings of 6th International Conference on Distributed Multimedia Systems (DMS '99), Aizu, Japan, July 1999.

InstantDB (1999), "InstantDB – A Java Database Engine", URL: http://www.instantdb.co.uk.

ISO (1997): "IT – Open Systems Interconnection, Data Management and Open Distributed Process. Reference Model of Open Distributed Processing. ODP Trading Function. Part 1: Specification. IS13235-1, 1997.

Kutvonen L. (1997), "Management of Application Federations", Proceedings of International IFIP Workshop on Distributed Applications and Interoperable Systems (DAIS '97), pp 33-46, Chapman and Hall, Oct 1997.

OMG (1998): "The Common Object Request Broker: Architecture and Specification. (Revision 2.2), 1998. URL: http://www.omg.org.

Richman A. and Hoang D. (1995a): "Trader Interoperability: Why Two Models are better than one", Proceedings of 10th International Symposium on Computer and Information Science (ISCIS '95), Turkey, 1995.

Richman A. and Hoang D. (1995b): "Accomplishing Distributed Traders utilising the X. 500 Directory", Proceedings of 2nd IEEE Malaysia International Conference on Communications (MICC '95), Malaysia, Nov 1995.

Waugh A. and Bearman M. (1995): "Designing an ODP Trader Implementation using X. 500", Proceedings of International Conference on Open Distributed Processing, Brisbane, Feb. 1995.

Issues Concerning the Development of a CORBA Trading Service

(Andrew K Lui and David J Miron)

(DSTO-TR-0841)

DISTRIBUTION LIST

	Number of Copies
AUSTRALIA	
DEFENCE ORGANISATION	
Task sponsor:	
DGC3ID	1
S&T Program	
Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Corporate Management)	
Director General Science Policy Development	1
Counsellor, Defence Science, London	Doc Control Sheet
Counsellor, Defence Science, Washington	Doc Control Sheet
Scientific Adviser to MRDC Thailand	Doc Control Sheet
Scientific Adviser - Policy and Command	1
Navy Scientific Adviser	1 copy of Doc Control Sheet and 1 distribution list
Scientific Adviser - Army	Doc Control Sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1
Aeronautical & Maritime Research Laboratory	
Director	1
Electronics and Surveillance Research Laboratory	
Director	1 copy of Doc Control Sheet and 1 distribution list
Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Research Leader Advanced Computer Capabilities	Doc Control Sheet
Research Leader Joint Systems	1
Head, Information Warfare Studies Group	Doc Control Sheet
Head, Software Systems Engineering Group	Doc Control Sheet
Head, Year 2000 Project	Doc Control Sheet
Head, Trusted Computer Systems Group	Doc Control Sheet
Head, Advanced Computer Capabilities Group	Doc Control Sheet
Head, Systems Simulation and Assessment Group	Doc Control Sheet

Head, C3I Operational Analysis Group	Doc Control Sheet
Head, Information Management and Fusion Group	1
Head, Human Systems Integration Group	Doc Control Sheet
Head, C2 Australian Theatre	1
Head, Information Architectures Group	1
Head, Distributed Systems Group	Doc Control Sheet
Head C3I Systems Concepts Group	1
Head, Organisational Change Group	Doc Control Sheet
Andrew K Lui (Author)	1
David J Miron (Author)	1
Publications and Publicity Officer, ITD/EOITD	1 shared copy
DSTO Library and Archives	
Library Fishermans Bend	1
Library Maribyrnong	1
Library Salisbury	2
Australian Archives	1
Library, MOD, Pyrmont	Doc Control Sheet
US Defense Technical Information Center	2
UK Defence Research Information Centre	2
Canada Defence Scientific Information Service	1
NZ Defence Information Centre	1
National Library of Australia	1
Capability Development Division	
Director General Maritime Development	Doc Control Sheet
Director General Land Development	Doc Control Sheet
Director General C3I Development	Doc Control Sheet
Director General Aerospace Development	Doc Control Sheet
Navy	
SO (Science), Director of Naval Warfare, Maritime Headquarters Annex, Garden Island, NSW 2000	Doc Control Sheet
Army	
ABCA Standardisation Officer, Puckapunyal	4
Intelligence Program	
DGSTA Defence Intelligence Organisation	1
Corporate Support Program	
OIC TRS Defence Regional Library, Canberra	1
Universities and Colleges	
Australian Defence Force Academy	1
Library	1
Head of Aerospace and Mechanical Engineering	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1
Serials Librarian Deakin University	1

Other Organisations	
NASA (Canberra)	1
AGPS 1	
State Library of South Australia	1
Parliamentary Library, South Australia	1
OUTSIDE AUSTRALIA	
Abstracting and Information Organisations	
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts US	1
Documents Librarian, The Center for Research Libraries, US	1
Information Exchange Agreement Partners	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	5
Total number of copies:	**58**

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
2. TITLE Issues Concerning the Development of a CORBA Trading Service	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Andrew K Lui and David J Miron	5. CORPORATE AUTHOR Electronics and Surveillance Research Laboratory PO Box 1500 Salisbury SA 5108 Australia

6a. DSTO NUMBER DSTO-TR-0841	6b. AR NUMBER AR-011-019	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE July 1999

8. FILE NUMBER n/a	9. TASK NUMBER 99/004	10. TASK SPONSOR DGC3ID	11. NO. OF PAGES 26	12. NO. OF REFERENCES 9

13. DOWNGRADING/DELIMITING INSTRUCTIONS n/a	14. RELEASE AUTHORITY Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS

Computer architecture
Object-oriented system architecture
Command control communications and intelligence

19. ABSTRACT

A core feature of the Experimental Command Communication Control Intelligence Technology Environment (EXC3ITE) is its support for distributed applications. A key feature of distributed applications is that the components of the applications can be developed and managed distributedly. At run time, the applications are dynamically composed using the components distributed in a networked environment. The Location Service is a critical enabling technology of distributed applications. It allows components to discover other collaborative components during run time. The Trading Service is a more advanced variant of the Location Service. This work focuses on the design and development of a trader running on CORBA. In particular an implementation of a trader using Java is presented, in addition to discussions on development issues and future research directions on traders.

Page classification: UNCLASSIFIED